Available online at www.sciencedirect.com
ScienceDirect
Discrete Optimization 4 (2007) 349–359
DISCRETE OPTIMIZATION
www.elsevier.com/locate/disopt

A strongly polynomial algorithm for line search in submodular polyhedra

Kiyohito Nagano

Department of Mathematical Informatics, Graduate School of Information Science and Technology, University of Tokyo, Bunkyo-ku, Tokyo 113-8656, Japan

Received 16 June 2004; received in revised form 2 May 2005; accepted 21 September 2007
Available online 31 October 2007

Abstract

A submodular polyhedron is a polyhedron associated with a submodular function. This paper presents a strongly polynomial time algorithm for line search in submodular polyhedra with the aid of a fully combinatorial algorithm for submodular function minimization. The algorithm is based on the parametric search method proposed by Megiddo.

Keywords: Submodular function; Network flow; Exchange capacity

1. Introduction

Let U be a finite nonempty set. A function ρ defined on 2^U is *submodular* if

$$\rho(X) + \rho(Y) \geq \rho(X \cup Y) + \rho(X \cap Y), \quad \forall X, Y \subseteq U. \tag{1}$$

Iwata, Fleischer and Fujishige [13] and Schrijver [21] independently presented combinatorial, strongly polynomial time algorithms for submodular function minimization. Iwata [11] presented a fully combinatorial strongly polynomial time algorithm, which uses only additions, subtractions, comparisons, and the oracle calls for function values.

For a vector $x \in \mathbf{R}^U$ and $u \in U$, we denote by $x(u)$ the component of x on u. For a submodular function $\rho : 2^U \to \mathbf{R}$ with $\rho(\emptyset) = 0$, the *submodular polyhedron* $\mathbf{P}(\rho)$ and the *base polyhedron* $\mathbf{B}(\rho)$ are defined by

$$\mathbf{P}(\rho) = \{x \in \mathbf{R}^U \mid x(X) \leq \rho(X)(\forall X \subseteq U)\}, \tag{2}$$

$$\mathbf{B}(\rho) = \{x \in \mathbf{R}^U \mid x \in \mathbf{P}(\rho), x(U) = \rho(U)\}, \tag{3}$$

where $x(X) = \sum_{u \in X} x(u)$.

Let V be a finite nonempty set with $|V| = n$ and let $f : 2^V \to \mathbf{R}$ be a submodular function with $f(\emptyset) = 0$. In this paper we consider the following problem:

Problem Line Search in Submodular Polyhedra (LSSP)

Instance: A submodular function $f : 2^V \to \mathbf{R}$ with $f(\emptyset) = 0$, a starting point $x_0 \in \mathbf{P}(f)$ and a direction vector $a \in \mathbf{R}^V$.

Task: Find $t^* = \max\{t \in \mathbf{R} \mid x_0 + ta \in \mathbf{P}(f)\}$.

E-mail address: kiyohito_nagano@mist.i.u-tokyo.ac.jp.

1572-5286/$ - see front matter
doi:10.1016/j.disopt.2007.09.002

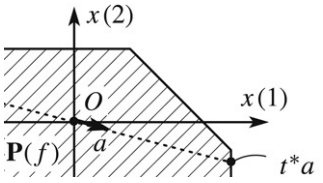

Fig. 1. Problem LSSP ($n = 2$).

Problem LSSP is a basic problem, but it is previously unknown if Problem LSSP can be solved in strongly polynomial time. We denote the set of nonpositive real numbers by $\mathbf{R}_-$, the set of nonnegative real numbers by $\mathbf{R}_+$. If $a \in \mathbf{R}_-^V$, Problem LSSP does not have an optimal solution. Hence throughout we assume that $a \notin \mathbf{R}_-^V$. We can assume $f(X) \geq 0$, $\forall X \subseteq V$, and $x_0 = \mathbf{0}$, by resetting $f(X) := f(X) - x_0(X)$ for all $X \subseteq V$. So throughout we assume that f is nonnegative, $f(\emptyset) = 0$ and $x_0 = \mathbf{0}$. An example of Problem LSSP is illustrated in Fig. 1.

The *lexicographically optimal base problem*, which was introduced by Fujishige [6], is a generalization of the lexicographically optimal flow problem considered by Megiddo [14]. Fujishige [6] (also see [7, Section 9.2]) showed that the lexicographically optimal base can be obtained by repeatedly solving Problem LSSP with $a \geq \mathbf{0}$ at most n times.

Let us consider the following problem, which is a special case of Problem LSSP:

Problem Line Search in Base Polyhedra (LSBP)

Instance: A submodular function $f : 2^V \to \mathbf{R}$ with $f(\emptyset) = 0$, a starting point $x_0 \in \mathbf{B}(f)$ and a direction vector $a \in \mathbf{R}^V$ with $a(V) = 0$.

Task: Find $t^* = \max\{t \in \mathbf{R} \mid x_0 + ta \in \mathbf{B}(f)\}$ $(= \max\{t \in \mathbf{R} \mid x_0 + ta \in \mathbf{P}(f)\})$.

As is the case with Problem LSSP, it is previously unknown if Problem LSBP can be solved in strongly polynomial time. We will see some problems associated with Problem LSBP in the following paragraphs.

Hartvigsen [9,10] addressed a *constrained submodular optimization problem*:

$$\max\left\{\sum_{v\in V} w(v)x(v) \mid x \in \mathbf{B}(f), \sum_{v\in V} b_i(v)x(v) = d_i \ (i = 1, \ldots, p)\right\},$$

where p is a nonnegative integer, $w \in \mathbf{R}^V$, and $b_i \in \mathbf{R}^V$, $d_i \in \mathbf{R}$ for each $i = 1, \ldots, p$. Hartvigsen [10] showed that the constrained submodular optimization problem can be solved in strongly polynomial time for *fixed* values of p. Problem LSBP is a special case of the constrained submodular optimization problem such that the feasible set has dimension 1 (or 0), that is, $p = \mathrm{O}(n)$. Thus Hartvigsen's result applied to Problem LSBP does not lead to a strongly polynomial time algorithm.

For each $v \in V$, let $\chi_v \in \mathbf{R}^V$ be the characteristic vector that has value 1 on v and 0 elsewhere. As an instance of Problem LSBP, if $a = \chi_v - \chi_{v'}$ for $v, v' \in V$, $v \neq v'$, the optimal value $t^* = \max\{t \in \mathbf{R} \mid x_0 + t(\chi_v - \chi_{v'}) \in \mathbf{B}(f)\}$ is said to be an *exchange capacity* (see, for example, Fujishige [7]) and can be computed directly by a submodular function minimization algorithm. So Problem LSBP can be interpreted as a problem of computing a *generalized exchange capacity*. In many algorithms for optimization problems associated with submodular functions, e.g. submodular flow problems, we compute exchange capacities iteratively. Approaches using generalized exchange capacities may provide good algorithms for problems associated with submodular functions.

The problems of finding maximum flow values in capacitated networks can be reduced to Problem LSBP. Consider a network $\mathcal{N}$ with a directed graph $G = (V, E)$, where V is a vertex set and E is an arc set, and with a nonnegative capacity vector $c \in \mathbf{R}^E$. For $X \subseteq V$, we denote the arc set $\{e = (v, v') \in E \mid v \in X,\ v' \in V \setminus X\}$ by $\delta(X)$. We define a function $\kappa_c : 2^V \to \mathbf{R}$ as $\kappa_c(X) = c(\delta(X))(X \subseteq V)$. This function κ_c is called a *cut function* and it is a nonnegative submodular function with $\kappa_c(\emptyset) = \kappa_c(V) = 0$. A vector $x \in \mathbf{R}^V$ is said to be *feasible* for $\mathcal{N}$ if there exists a vector $y \in \mathbf{R}^E$ such that

$$\mathbf{0} \leq y \leq c, \quad \text{and} \quad \sum\{y(e) \mid e \in \delta(\{v\})\} - \sum\{y(e) \mid e \in \delta(V \setminus \{v\})\} = x(v), \quad \forall v \in V, \tag{4}$$

that is, there exists a flow y in network $\mathcal{N}$ which satisfies capacity constraints w.r.t. c and supply constraints w.r.t. x.

Maximum r–s flows. Let $r, s \in V$, $r \neq s$ and we consider finding the maximum flow value from r to s. The maximum r–s flow value is $t^* = \max\{t \in \mathbf{R} \mid t(\chi_r - \chi_s)$ is feasible for $\mathcal{N}\}$. Gale [8] showed that the set

$\{x \in \mathbf{R}^V \mid x \text{ is feasible for } \mathcal{N}\}$ is equal to the base polyhedron $\mathbf{B}(\kappa_c)$. Therefore we can find the maximum r–s flow value by solving Problem LSBP.

Maximum $\mathbf{w}$*-proportional flows.* Let $S^+, S^- \subseteq V$, $S^+ \cap S^- = \emptyset$ and let $w \in \mathbf{R}^V$ be a vector which satisfies

$$\sum\{w(v) \mid v \in S^+\} = 1, \qquad \sum\{w(v) \mid v \in S^-\} = -1 \quad \text{and} \quad \begin{cases} w(v) > 0 & (v \in S^+), \\ w(v) < 0 & (v \in S^-), \\ w(v) = 0 & (v \in V \setminus (S^+ \cup S^-)). \end{cases}$$

For $t \in \mathbf{R}$, we say that $y \in \mathbf{R}^E$ is a w-proportional flow with flow value t if y satisfies (4) w.r.t. $x = tw$. The maximum w-proportional flow value is $t^* = \max\{t \in \mathbf{R} \mid tw \in \mathbf{B}(\kappa_c)\}$ and this is the optimal value of Problem LSBP.

The Newton method (Section 4) is a simple approach to Problem LSSP. If $a \in \mathbf{R}^V_+ \setminus \{\mathbf{0}\}$, it is shown that the number of iterations of the Newton method is at most $n + 1$ and Problem LSSP can be solved in strongly polynomial time. (See Fujishige [7, Section 7.2], Fleischer and Iwata [5].) If $a \in \mathbf{R}^V$ and $a \notin \mathbf{R}^V_+$, however, only a weakly polynomial running time bound is given, and it is left open to verify if the Newton method for Problem LSSP runs in strongly polynomial time.

In this paper, we propose an algorithm for Problem LSSP, which is quite different from the Newton method. The algorithm uses a fully combinatorial algorithm for submodular function minimization [11,12], within the framework of the parametric search method proposed by Megiddo [15,16]. It solves Problem LSSP in strongly polynomial time.

From the definition of a submodular polyhedron (2), it is easy to see that the optimal value t^* of Problem LSSP is equal to $\min\{f(X)/a(X) \mid X \subseteq V, a(X) > 0\}$. So Problem LSSP can be regarded as a minimum-ratio problem. Using the same technique as the algorithm for Problem LSSP, we also show that a minimum-ratio problem which is a generalization of Problem LSSP can be solved in strongly polynomial time. The paper is organized as follows. In Section 2, we provide preliminaries for the following sections. Section 3 discusses algorithms for submodular function minimization. In Section 4, we describe the Newton method using an algorithm for submodular function minimization as a subroutine. Section 5 presents a strongly polynomial time algorithm for Problem LSSP using a fully combinatorial algorithm for submodular function minimization within the framework of the parametric search method proposed by Megiddo, and Section 6 gives a strongly polynomial time algorithm for a minimum-ratio problem which is a generalization of Problem LSSP.

2. Preliminaries

2.1. Definitions and basic properties

Let U be a finite nonempty set. A family $\mathcal{D} \subseteq 2^U$ is said to be a *ring family* if it satisfies $X, Y \in \mathcal{D} \Rightarrow X \cup Y, X \cap Y \in \mathcal{D}$. Of course 2^U is a ring family. Let $\mathcal{D} \subseteq 2^U$ be a ring family. A function $\rho : \mathcal{D} \to \mathbf{R}$ is called *submodular* if

$$\rho(X) + \rho(Y) \geq \rho(X \cup Y) + \rho(X \cap Y), \quad \forall X, Y \in \mathcal{D}. \tag{5}$$

A function $\rho : \mathcal{D} \to \mathbf{R}$ is called *supermodular* if $-\rho$ is submodular, and a function $\rho : \mathcal{D} \to \mathbf{R}$ is called *modular* if ρ is submodular and supermodular. For a modular function $\rho : \mathcal{D} \to \mathbf{R}$, ρ can be expressed as $\rho(X) = b_0 + b(X)$, $\forall X \in \mathcal{D}$, using some $b_0 \in \mathbf{R}$ and $b \in \mathbf{R}^U$.

Let $\mathcal{D} \subseteq 2^U$ be a ring family and let $\rho : \mathcal{D} \to \mathbf{R}$ be a submodular function. Let $\arg\min \rho \subseteq \mathcal{D}$ denote a family of all the minimizers of ρ. It is not difficult to see that $\arg\min \rho$ forms a ring family using the submodularity of ρ. As $\arg\min \rho$ is closed under union and intersection, there exists the *minimal minimizer* $X_{\min} = \bigcap \arg\min \rho$ and the *maximal minimizer* $X_{\max} = \bigcup \arg\min \rho$.

Let $\rho : 2^U \to \mathbf{R}$ be a submodular function with $\rho(\emptyset) = 0$ and let $x \in \mathbf{P}(\rho)$. A subset $X \subseteq U$ is said to be x*-tight* w.r.t. ρ if $x(X) = \rho(X)$. We denote the family of x-tight sets w.r.t. ρ by $\mathcal{D}_\rho(x)$. Namely, $\mathcal{D}_\rho(x) = \{X \subseteq U \mid x(X) = \rho(X)\}$. For any $y \in \mathbf{R}^U$, a function $\rho_y : 2^U \to \mathbf{R}$ defined by $\rho_y(X) = \rho(X) - y(X) (X \subseteq V)$ is obviously a submodular function and $\rho_y(\emptyset) = 0$. As $x \in \mathbf{P}(\rho)$, $\rho_x(\emptyset) = 0$ and $\rho_x(X) \geq 0$, $\forall X \subseteq U$. This implies

$$X \in \mathcal{D}_\rho(x) \iff X \in \arg\min \rho_x.$$

So $\mathcal{D}_\rho(x) = \arg\min \rho_x$, therefore $\mathcal{D}_\rho(x)$ forms a ring family. Note that $\emptyset \in \mathcal{D}_\rho(x)$.

Let $\mathcal{D} \subseteq 2^U$ be a ring family. Now we assume $\{\emptyset, U\} \subseteq \mathcal{D}$. Although the cardinality of $\mathcal{D}$ may be exponentially large in general, $\mathcal{D}$ can always be represented by a directed graph with at most $|U|$ vertices, that is, $O(|U|^2)$ size.

For a directed graph $G = (U, A)$, we call $X \subseteq U$ a *closure* of G if $(u, u') \in A$ and $u \in X$ implies $u' \in X$. It is known that there exists a directed graph $G_{\mathcal{D}} = (U, A_{\mathcal{D}})$ such that $\mathcal{D}$ is the family of closures of G. For example, if $G_{\mathcal{D}} = (U, A_{\mathcal{D}})$ is a directed graph with

$$A_{\mathcal{D}} = \left\{(u, u') | u, u' \in U, u \neq u', u' \in \bigcap \{X \mid u \in X \in \mathcal{D}\}\right\}, \tag{6}$$

then $\mathcal{D} = \{X \mid X \subseteq U \text{ is a closure of } G_{\mathcal{D}}\}$ (see, e.g., [7, Section 3.2]). We say that $G_{\mathcal{D}}$ is a *directed graph representation* of $\mathcal{D}$. We decompose $G_{\mathcal{D}}$ into strongly connected components $\{\Gamma_{\mathcal{D}}(s) \mid s \in S\}$, where $\Gamma_{\mathcal{D}}(s)$ is a subset of U for each $s \in S$. Let $\mathcal{G}_{\mathcal{D}} = (S, F_{\mathcal{D}})$ be a directed acyclic graph determined by $G_{\mathcal{D}}$ in a natural way. For each $T \subseteq S$, we define $\Gamma_{\mathcal{D}}(T) = \bigcup_{s \in T} \Gamma_{\mathcal{D}}(s) \subseteq U$. Now $\mathcal{D} = \{\Gamma_{\mathcal{D}}(T) | T \subseteq S \text{ is a closure of } \mathcal{G}_{\mathcal{D}}\}$. We say that $(\Gamma_{\mathcal{D}}, \mathcal{G}_{\mathcal{D}})$ is a *contracted directed graph representation* of $\mathcal{D}$. A ring family $\mathcal{D}$ is called *simple* if $G_{\mathcal{D}}$ is acyclic. For a simple ring family $\mathcal{D}$, we can identify $G_{\mathcal{D}}$ with $(\Gamma_{\mathcal{D}}, \mathcal{G}_{\mathcal{D}})$. For a ring family $\mathcal{D}$, we consider the case when $\{\emptyset, U\} \not\subseteq \mathcal{D}$. Let $U_{\mathsf{res}} = \bigcup \mathcal{D} \setminus \bigcap \mathcal{D}$ and $\mathcal{D}_{\mathsf{res}} = \{X \setminus \bigcap \mathcal{D} \mid X \in \mathcal{D}\}$. Note that $\mathcal{D}_{\mathsf{res}}$ is a ring family with $\{\emptyset, U_{\mathsf{res}}\} \subseteq \mathcal{D}_{\mathsf{res}} \subseteq 2^{U_{\mathsf{res}}}$. We define $G_{\mathcal{D}} := G_{\mathcal{D}_{\mathsf{res}}}$, $\mathcal{G}_{\mathcal{D}} := \mathcal{G}_{\mathcal{D}_{\mathsf{res}}}$, and $\Gamma_{\mathcal{D}} := \Gamma_{\mathcal{D}_{\mathsf{res}}}$. We say that $(\bigcap \mathcal{D}, \bigcup \mathcal{D}, G_{\mathcal{D}})$ is a directed graph representation of $\mathcal{D}$, and that $(\bigcap \mathcal{D}, \bigcup \mathcal{D}, (\Gamma_{\mathcal{D}}, \mathcal{G}_{\mathcal{D}}))$ is a contracted directed graph representation of $\mathcal{D}$. All the information about $\mathcal{D}$ is equivalent to $(\bigcap \mathcal{D}, \bigcup \mathcal{D}, G_{\mathcal{D}})$ or $(\bigcap \mathcal{D}, \bigcup \mathcal{D}, (\Gamma_{\mathcal{D}}, \mathcal{G}_{\mathcal{D}}))$.

2.2. Optimality conditions for problem LSSP

As an instance of Problem LSSP, w.l.o.g., we assume that f is nonnegative, $f(\emptyset) = 0$, $x_0 = \mathbf{0}$, and $a \notin \mathbf{R}^V_-$. We explain that the optimal value t^* of Problem LSSP is nonnegative and finite. The optimal value is by definition

$$t^* = \max\{t \mid ta \in \mathbf{P}(f)\}. \tag{7}$$

Since $\mathbf{0} \in \mathbf{P}(f)$, t^* is nonnegative. Let $A \subseteq V$ be a subset which satisfies $a(A) > 0$. If $t > f(A)/a(A)$, then $ta(A) > f(A)$ and hence $ta \notin \mathbf{P}(f)$. So $t^* \leq f(A)/a(A)$ and t^* is finite.

For any $t \in \mathbf{R}$ we consider deciding whether $ta \in \mathbf{P}(f)$ or $ta \notin \mathbf{P}(f)$. Since, for any $x \in \mathbf{R}^V$, $f(\emptyset) - x(\emptyset) = 0$, we have

$$x \in \mathbf{P}(f) \iff \min\{f_x(X) \mid X \subseteq V\} = 0,$$

and if x can be represented as ta, using $ta(\emptyset) = 0$,

$$\begin{aligned} ta \in \mathbf{P}(f) &\iff \min\{f_{ta}(X) \mid X \subseteq V\} = 0, \\ ta \notin \mathbf{P}(f) &\iff \min\{f_{ta}(X) \mid X \subseteq V\} < 0. \end{aligned} \tag{8}$$

So we can decide whether $ta \in \mathbf{P}(f)$ or $ta \notin \mathbf{P}(f)$ by minimizing f_{ta}.

Now, let us consider the optimality condition for Problem LSSP. For $t \geq 0$, we consider the conditions of "$t < t^*$", "$t = t^*$" and "$t > t^*$". See Fig. 2 to understand each condition intuitively. Note that $t = t^*$ and ta in the boundary of $\mathbf{P}(f)$ are not equivalent (see Ex. 1. 2 and Ex. 1. 3 in Fig. 2).

Eq. (7) directly implies that

$$\begin{aligned} t = t^* &\iff ta \in \mathbf{P}(f), \quad \forall \varepsilon > 0\ (t + \varepsilon)a \notin \mathbf{P}(f), \\ &\iff ta \in \mathbf{P}(f), \quad \exists X \subseteq V \text{ such that } \forall \varepsilon > 0\ \varepsilon a(X) > f_{ta}(X)\ (\geq 0), \\ &\iff ta \in \mathbf{P}(f), \quad \exists X \in \mathcal{D}_f(ta) \text{ such that } a(X) > 0, \\ &\iff ta \in \mathbf{P}(f), \quad \max\{a(X) \mid X \in \mathcal{D}_f(ta)\} > 0. \end{aligned} \tag{9}$$

For $ta \in \mathbf{P}(f)$, $\mathcal{D}_f(ta)$ always includes $\emptyset$, so $\max\{a(X) \mid X \in \mathcal{D}_f(ta)\} \geq 0$. Thus using (8) and (9), we obtain the following conditions for t^* and any $t \geq 0$:

$$\begin{aligned} t < t^* &\iff \begin{cases} \min\{f_{ta}(X) \mid X \subseteq V\} = 0, \\ \max\{a(X) \mid X \in \mathcal{D}_f(ta)\} = 0, \end{cases} \\ t = t^* &\iff \begin{cases} \min\{f_{ta}(X) \mid X \subseteq V\} = 0, \\ \max\{a(X) \mid X \in \mathcal{D}_f(ta)\} > 0, \end{cases} \\ t > t^* &\iff \min\{f_{ta}(X) \mid X \subseteq V\} < 0. \end{aligned} \tag{10}$$

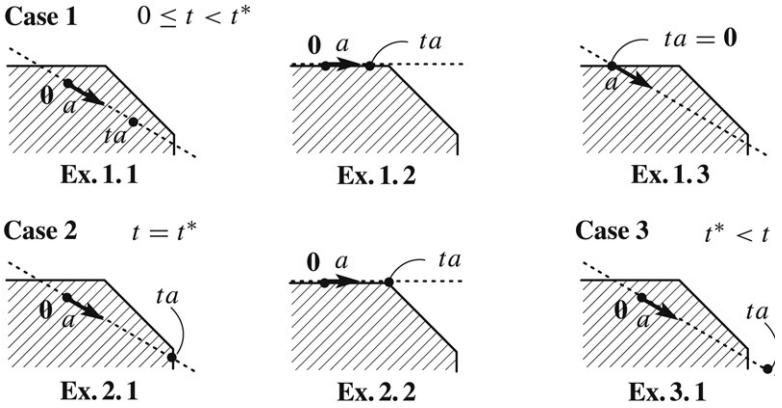

Fig. 2. Relation between t and t^*.

3. Submodular function minimization

3.1. Finding a minimizer of a submodular function

Let U be a finite nonempty set and let $\rho : 2^U \to \mathbf{R}$ be any submodular function. We assume that for any given $X \subseteq U$ a function value $\rho(X)$ can be acquired by an *oracle call*. Let $\gamma(\rho)$ denote the upper bound on the time to compute the function value of ρ. An algorithm for submodular function minimization is said to be a strongly polynomial time algorithm if for any submodular function $\rho : 2^U \to \mathbf{R}$ the total number of oracle calls for function evaluation and *arithmetic operations*, that is, additions, subtractions, multiplications, divisions and comparisons, is bounded by some polynomial in $|U|$. Combinatorial strongly polynomial time algorithms for submodular function minimization are given independently by Iwata, Fleischer and Fujishige (IFF) [13] and Schrijver [21]. Iwata [12] described an improved variant of the IFF algorithm and this algorithm achieves the best known bound on the running time, $O((|U|^6 \log |U|) \cdot \gamma(\rho) + |U|^7 \log |U|)$. Let Algorithm SFM be an algorithm which finds a minimizer of a submodular function $\rho : 2^U \to \mathbf{R}$ with $O(\mathcal{T}^{\mathsf{O}}(|U|))$ oracle calls for function evaluation and $O(\mathcal{T}^{\mathsf{A}}(|U|))$ arithmetic operations where $\mathcal{T}^{\mathsf{O}}(|U|)$ and $\mathcal{T}^{\mathsf{A}}(|U|)$ are some polynomials in $|U|$. Let $\mathcal{T}(|U|, \gamma(\rho)) = \mathcal{T}^{\mathsf{O}}(|U|) \cdot \gamma(\rho) + \mathcal{T}^{\mathsf{A}}(|U|)$.

An algorithm for submodular function minimization is said to be a *fully combinatorial* strongly polynomial time algorithm if the total number of oracle calls for function evaluation of ρ and *fully combinatorial operations*, that is, additions, subtractions and comparisons, is bounded by some polynomial in $|U|$. Iwata [11] presented a fully combinatorial strongly polynomial time algorithm for submodular function minimization as a variant of the IFF algorithm, and later, Iwata [12] described an improved algorithm, which runs in $O(|U|^8 \log^2 |U| \cdot \gamma(\rho))$ time. Let Algorithm FC-SFM be a fully combinatorial algorithm which finds a minimizer of a submodular function $\rho : 2^U \to \mathbf{R}$ with $O(\mathcal{T}^{\mathsf{O}}_{\mathsf{FC}}(|U|))$ oracle calls for function evaluation of ρ and $O(\mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(|U|))$ fully combinatorial operations, where $\mathcal{T}^{\mathsf{O}}_{\mathsf{FC}}(|U|)$ and $\mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(|U|)$ are some polynomials in $|U|$. Let $\mathcal{T}_{\mathsf{FC}}(|U|, \gamma(\rho)) = \mathcal{T}^{\mathsf{O}}_{\mathsf{FC}}(|U|) \cdot \gamma(\rho) + \mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(|U|)$.

3.2. Constructing all the minimizers of a submodular function

Let $\rho : 2^U \to \mathbf{R}$ be a submodular function and X_{min}, X_{max} be the minimal, maximal minimizer of ρ, respectively. We are interested in constructing $\arg\min \rho$, that is, finding X_{min}, X_{max}, and $G_{\arg\min \rho}$ (or $(\Gamma_{\arg\min \rho}, \mathcal{G}_{\arg\min \rho})$).

A (fully combinatorial) algorithm which computes X_{min}, X_{max} and $G_{\arg\min \rho}$ can be designed by using any (fully combinatorial) algorithm which finds the minimal minimizer of a submodular function $|U|$ times. For each $u \in U$, let $\rho_u : 2^{U\setminus\{u\}} \to \mathbf{R}$ be a submodular function defined by $\rho_u(X) = \rho(X \cup \{u\})$ $(X \subseteq U \setminus \{u\})$. We compute the minimal minimizer M_u of f_u for each $u \in U$, and put $\alpha = \min\{\rho(\emptyset), \ \min\{\rho(M_u) \mid u \in U\}\}$. Note that α is equal to the minimum value of ρ. It is easy to see that

$$X_{\mathsf{min}} = \bigcap \{M_u \mid u \in U, f(M_u) = \alpha\} \quad \text{and} \quad X_{\mathsf{max}} = \bigcup \{M_u \mid u \in U, f(M_u) = \alpha\}.$$

We can obtain $G_{\arg\min\rho}$ using (6) and the information about $\{M_u \mid u \in U, f(M_u) = \alpha\}$. It is known that the IFF algorithm and its variants always find the maximal minimizer of a submodular function. For a submodular function $\rho : 2^U \to \mathbf{R}$, a function $\rho^\sharp : 2^U \to \mathbf{R}$ defined by $\rho^\sharp(X) = \rho(U \setminus X)$ $(X \subseteq V)$ is also submodular. So any algorithm finding the maximal (minimal) minimizer of a submodular function can be transformed to an algorithm finding the minimal (maximal) minimizer of a submodular function. Therefore we can construct $\arg\min\rho$ using the IFF algorithm (or its variants) $|U|$ times. The minimal minimizer of a submodular function can be easily computed using any submodular function minimization algorithm $|U|+1$ times. (See, e.g., Note 10.12. in [17].) Hence $\arg\min\rho$ can be constructed using any submodular function minimization algorithm $O(|U|^2)$ times.

As for currently known combinatorial, strongly polynomial time algorithms [11–13,21], we can do much better than that: using each one of them, we can construct $\arg\min\rho$ in the same asymptotic running time as a single computation of the original algorithm. Now we explain how to achieve this.

As resetting $\rho(X) := \rho(X) - \rho(\emptyset)$ $(X \subseteq U)$ does not change the problem, we can assume $\rho(\emptyset) = 0$. For $x \in \mathbf{R}^U$, we define $x^- \in \mathbf{R}^U$ by $x^-(u) = \min\{0, x(u)\}$ for $u \in U$, and define $\mathrm{supp}^-(x), \mathrm{supp}^+(x) \subseteq U$ by $\{u \in U \mid x(u) < 0\}$, $\{u \in U \mid x(u) > 0\}$ respectively. For any $x \in \mathbf{B}(\rho)$ and any $X \subseteq U$, we have $x^-(U) \le x(X) \le \rho(X)$, and the vector reduction theorem on polymatroids due to Edmonds [4] immediately implies

$$\max\{x^-(U) \mid x \in \mathbf{B}(\rho)\} = \min\{\rho(X) \mid X \subseteq U\}. \tag{11}$$

So, for a maximizer x' of the left-hand side of (11), we have

$$\arg\min\rho = \{X \mid X \in \mathcal{D}_\rho(x'), \mathrm{supp}^-(x') \subseteq X \subseteq U \setminus \mathrm{supp}^+(x')\}. \tag{12}$$

An extreme point of a base polyhedron is said to be an *extreme base*. The following theorem plays an important role for the construction of $\arg\min\rho$.

Theorem 3.1 (*Bixby, Cunningham and Topkis [2]*). *Let $\rho : 2^U \to \mathbf{R}$ be a submodular function with $\rho(\emptyset) = 0$, and let b be an extreme base of $\mathbf{B}(\rho)$.*

(1) *$\mathcal{D}_\rho(b)$ includes $\{\emptyset, U\}$ and is a simple ring family.*
(2) *A directed graph representation of $\mathcal{D}_\rho(b)$ can be constructed in $O(|U|^2 \cdot \gamma(\rho))$ time.* □

If a maximizer x' of the left-hand side of (11) is given as a convex combination of k extreme bases, then using (12) and Theorem 3.1 we can construct $\arg\min\rho$ in $O(k\,|U|^2 \cdot \gamma(\rho))$ time. (Refer to Note 10.11. in [17] for details.) Schrijver's algorithm finds not only a minimizer of the right-hand side of (11), but also a maximizer of the left-hand side of (11) which is represented as a convex combination of, we may assume, at most $|U|$ extreme points of $\mathbf{B}(\rho)$. Vygen [22] showed that Schrijver's algorithm runs in $O(|U|^7 \cdot \gamma(\rho) + |U|^8)$ time. So $\arg\min\rho$ can also be constructed in $O(|U|^7 \cdot \gamma(\rho) + |U|^8)$ time.

Unfortunately, the IFF algorithm and its variants do not find a maximizer of the left-hand side of (11). But we can overcome this difficulty. They use the same framework and we can construct $\arg\min\rho$ in the same way. In the algorithms, we maintain $Z, H \subseteq U$, a partition $\{\Gamma(s) \mid s \in S\}$ of $U \setminus (Z \cup H)$ and a directed acyclic graph $\mathcal{G} = (S, F)$ such that

⟨IFF-1⟩ $Z \subseteq X_{\min}, H \subseteq (U \setminus X_{\max})$,
⟨IFF-2⟩ for each $s \in S$ and $X \in \arg\min\rho$, $\Gamma(s) \subseteq X$ or $\Gamma(s) \cap X = \emptyset$,
⟨IFF-3⟩ for each arc $(s, s') \in F$ and $X \in \arg\min\rho$, $\Gamma(s) \subseteq X$ implies $\Gamma(s') \subseteq X$.

Note that $\{Z, H\} \cup \{\Gamma(s) \mid s \in S\}$ is a partition of U. Intuitively, in the algorithms, we update $(Z, U \setminus H, (\Gamma, \mathcal{G}))$ toward the construction of a contracted directed graph representation of $\arg\min\rho$, $(X_{\min}, X_{\max}, (\Gamma_{\arg\min\rho}, \mathcal{G}_{\arg\min\rho}))$. For each $T \subseteq S$, we define $\Gamma(T) = \bigcup_{s \in T} \Gamma(s)$. We define a function $\hat\rho : 2^S \to \mathbf{R}$ by

$$\hat\rho(T) = \rho(\Gamma(T) \cup Z) - \rho(Z) (T \subseteq S).$$

It is obvious that $\hat\rho$ is submodular and $\hat\rho(\emptyset) = 0$. We denote $\bigcap \arg\min\hat\rho$, $\bigcup \arg\min\hat\rho$ by $\hat X_{\min}$, $\hat X_{\max}$ respectively. By ⟨IFF-1⟩, ⟨IFF-2⟩ and the definition of $\hat\rho$, for each minimizer $T \subseteq S$ of $\hat\rho$, $\Gamma(T) \cup Z$ is minimizer of ρ, and, conversely, for each minimizer X of ρ there exists a minimizer T of $\hat\rho$ such that $X = \Gamma(T) \cup Z$. For $s \in S$, let $R(s) \subseteq S$ denote the set of vertices reachable from s in $\mathcal{G}$. In the algorithms, Z, H, Γ, and $\mathcal{G} = (S, F)$ finally satisfy the following

inequality:

$$\eta := \max\{\hat{\rho}(R(s)) - \hat{\rho}(R(s) \setminus \{s\}) \mid s \in S\} \leq 0.$$

We assume $\eta \leq 0$. As $\mathcal{G} = (S, F)$ is acyclic, there exists a linear order $\hat{L}$ on S in which for each $(s, s') \in F$, s' is a predecessor of s. Let $\hat{L} = (s_1, \ldots, s_{|S|})$ be such a linear order. We define $\hat{L}(s_j) = \{s_1, \ldots, s_j\} (j = 1, \ldots, |S|)$. By definition, for each $s \in S$, $R(s) \subseteq \hat{L}(s)$. Let $\hat{b} \in \mathbf{R}^S$ be a vector defined by

$$\hat{b}(s) = \hat{\rho}(\hat{L}(s)) - \hat{\rho}(\hat{L}(s) \setminus \{s\}) (s \in S).$$

Now $\hat{b}$ is an extreme base of $\mathbf{B}(\hat{\rho})$ (see Edmonds [4]). For each $s \in S$, we have, by the submodularity of $\hat{\rho}$, $\hat{b}(s) \leq \hat{f}(R(s)) - \hat{f}(R(s) \setminus \{s\}) \leq \eta \leq 0$, that is, $\hat{b} \leq \mathbf{0}$. Since $\hat{b} \in \mathbf{B}(\hat{\rho})$ and $\hat{b} \leq \mathbf{0}$, we have for any $T \subseteq S$

$$\hat{b}^-(S) = \hat{b}(S) = \hat{\rho}(S) \leq \hat{b}(T) \leq \hat{\rho}(T). \tag{13}$$

By (13), S is a minimizer of $\hat{\rho}$ and, of course, $\hat{X}_{\mathsf{max}} = S$. So $\Gamma(S) \cup Z$ is the maximal minimizer of ρ. The original algorithms output $U \setminus H = \Gamma(S) \cup Z$ as a minimizer of ρ and stop. We can construct $\arg\min \hat{\rho}$ using $\hat{b} \in \mathbf{B}(\hat{\rho})$ as follows. By (13) we have

$$\arg\min \hat{\rho} = \{T \mid T \in \mathcal{D}_{\hat{\rho}}(\hat{b}), \mathrm{supp}^-(\hat{b}) \subseteq T \subseteq S\}. \tag{14}$$

As $\hat{b}$ is an extreme base of $\mathbf{B}(\hat{\rho})$, we can easily obtain $G_{\mathcal{D}_{\hat{\rho}}(\hat{b})} = (S, A_{\mathcal{D}_{\hat{\rho}}(\hat{b})})$ by Theorem 3.1. It follows from (14) that $\hat{X}_{\mathsf{min}}$ is the set of vertices reachable from $\mathrm{supp}^-(\hat{b})$ in $G_{\mathcal{D}_{\hat{\rho}}(\hat{b})}$. Let G' be the subgraph of $G_{\mathcal{D}_{\hat{\rho}}(\hat{b})}$ induced by $S \setminus \hat{X}_{\mathsf{min}}$. Note that G' is acyclic. It is easy to see that $(\hat{X}_{\mathsf{min}}, S, G')$ is a directed graph representation of $\arg\min \hat{\rho}$. From the correspondence between $\arg\min \hat{\rho}$ and $\arg\min \rho$, we can construct $\arg\min \rho$ straightforward. Let $\Gamma' : S \setminus \hat{X}_{\mathsf{min}} \to 2^U$ be a set-valued function defined by $\Gamma'(s) = \Gamma(s)$ $(s \in S \setminus \hat{X}_{\mathsf{min}})$. Then $(X_{\mathsf{min}}, X_{\mathsf{max}}, (\Gamma', G'))$ is a contracted directed graph representation of $\arg\min \rho$ where $X_{\mathsf{min}} = \Gamma(\hat{X}_{\mathsf{min}}) \cup Z$ and $X_{\mathsf{max}} = \Gamma(S) \cup Z$. As a result, we can design a combinatorial algorithm which constructs $\arg\min \rho$ in $\mathrm{O}((|U|^6 \log |U|) \cdot \gamma(\rho) + |U|^7 \log |U|)$ time. Moreover, we can design a fully combinatorial algorithm which constructs $\arg\min \rho$ in $\mathrm{O}(|U|^8 \log^2 |U| \cdot \gamma(\rho))$ time.

Let Algorithm $\mathsf{SFM}_{\mathsf{am}}$ be some combinatorial strongly polynomial time algorithm which constructs all the minimizers of a submodular function $\rho : 2^U \to \mathbf{R}$, and let Algorithm $\mathsf{FC\text{-}SFM}_{\mathsf{am}}$ be some fully combinatorial strongly polynomial time algorithm which constructs all the minimizers of a submodular function $\rho : 2^U \to \mathbf{R}$. For simplicity we assume that the running time of $\mathsf{SFM}_{\mathsf{am}}$ is $\mathrm{O}(\mathcal{T}(|U|, \gamma(\rho)))$ and that of $\mathsf{FC\text{-}SFM}_{\mathsf{am}}$ is $\mathrm{O}(\mathcal{T}_{\mathsf{FC}}(|U|, \gamma(\rho)))$.

4. The Newton method for problem LSSP

In this section we describe the Newton method for Problem LSSP. The Newton method for Problem LSSP uses an algorithm for submodular function minimization as a subroutine. We define a function $h : \mathbf{R} \to \mathbf{R}$ as

$$h(t) = \min\{f_{ta}(X) \mid X \subseteq V\} = \min\{f(X) - ta(X) \mid X \subseteq V\}. \tag{15}$$

It is obvious that h is concave. As $f(\emptyset) = 0$ and $\mathbf{0} \in \mathbf{P}(f)$, $h(0) = 0$. Since $f_{ta}(\emptyset) = 0$ for any $t \in \mathbf{R}$, $h(t) \leq 0$ for any $t \in \mathbf{R}$. Using (7), (8) and (15), we have $t^* = \max\{t \in \mathbf{R} \mid h(t) = 0\}$. The graph of h is illustrated in Fig. 3 by a thick curve. For any $t \in \mathbf{R}$ we can obtain the value $h(t)$ by running $\mathsf{SFM}(f - ta)$. For simplicity, we assume $n = \mathrm{O}(\gamma(f))$ in the rest of the paper. (Remember that we reset $f(X) := f(X) - x_0(X)$ for all $X \subseteq V$ in Section 1.) Using this assumption, for any $x \in \mathbf{R}^V$, the function value of f_x can be acquired in $\mathrm{O}(\gamma(f))$ time. So $f - ta$ can be minimized in $\mathrm{O}(\mathcal{T}(n, \gamma(f)))$ time. The Newton method is described below. Fig. 3 illustrates the process of the algorithm.

The Newton method for Problem LSSP

Step 0: For $X_0 \subseteq V$ such that $a(X_0) > 0$, set $t_1 := f(X_0)/a(X_0)$ $(\geq t^*)$. Set $i := 1$.
Step 1: Obtain $X_i \subseteq V$ such that $h(t_i) = f(X_i) - t_i a(X_i)$ by running $\mathsf{SFM}(f - t_i a)$.
Step 2: If $h(t_i) = 0$, return $t^* := t_i$ and stop. If $h(t_i) < 0$ then set $t_{i+1} := f(X_i)/a(X_i)$ and $i := i + 1$. Go to Step 1.

As $h(t)$ has at most 2^n linear segments, the Newton method terminates in a finite number of iterations. If $a \in \mathbf{R}^V_+ \setminus \{\mathbf{0}\}$, it is known that the number of iterations of the Newton method for Problem LSSP is at most $n + 1$. (See Fujishige [7, Section 7.2], Fleischer and Iwata [5] for details.) But it is left open to verify if the Newton method for Problem

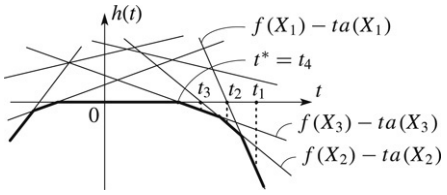

Fig. 3. The Newton method.

LSSP with arbitrary $a \in \mathbf{R}^V$ runs in strongly polynomial time. An analysis based on Radzik [19,20] gives a weakly polynomial bound on the number of iterations.

Theorem 4.1. *Let f be an integer-valued nonnegative submodular function with $f(\emptyset) = 0$ and a be an integer vector which satisfies $a \notin \mathbf{R}^V_-$. If $\max_{X \subseteq V} |f(X)| \leq U_1$, $\max_{X \subseteq V} |a(X)| \leq U_2$, then the Newton method for Problem* LSSP *runs in* $O(\log U_1 + \log U_2)$ *iterations.* □

5. A strongly polynomial algorithm

In this section we present a combinatorial strongly polynomial time algorithm for Problem LSSP. We use a fully combinatorial strongly polynomial algorithm for submodular function minimization [11,12] and the parametric search method proposed by Megiddo [15,16].

5.1. Framework

Later we will describe two procedures for Comparison with the Optimal Value t^*; Procedure COV and Procedure L-COV.

Procedure COV. For any given nonnegative value $t \geq 0$, we can tell whether "$t < t^*$", "$t = t^*$" or "$t > t^*$" by running $\mathsf{COV}(t)$ in $O(\mathcal{T}^{\mathsf{O}}_{\mathsf{COV}}(n) \cdot \gamma(f) + \mathcal{T}^{\mathsf{A}}_{\mathsf{COV}}(n))$ time, where $\mathcal{T}^{\mathsf{O}}_{\mathsf{COV}}(n)$ and $\mathcal{T}^{\mathsf{A}}_{\mathsf{COV}}(n)$ are some polynomials in n.

Procedure L-COV. For any given nonnegative value $t \geq 0$, once $ta(v)$ is computed for each $v \in V$, Procedure L-COV compares t to t^* with $O(\mathcal{T}^{\mathsf{O}}_{\text{L-COV}}(n))$ oracle calls for function evaluation of f, $O(\mathcal{T}^{\mathsf{FC}}_{\text{L-COV}}(n))$ comparisons, and $O(n\mathcal{T}^{\mathsf{O}}_{\text{L-COV}}(n) + \mathcal{T}^{\mathsf{FC}}_{\text{L-COV}}(n))$ additions and subtractions, where $\mathcal{T}^{\mathsf{O}}_{\text{L-COV}}(n)$ and $\mathcal{T}^{\mathsf{FC}}_{\text{L-COV}}(n)$ are some polynomials in n. As we assumed $n = O(\gamma(f))$ (see Section 4), the running time of $\text{L-COV}(t)$ is $O(\mathcal{T}^{\mathsf{O}}_{\text{L-COV}}(n) \cdot \gamma(f) + \mathcal{T}^{\mathsf{FC}}_{\text{L-COV}}(n))$. Moreover, if $t = t^*$, Procedure L-COV returns a subset $X \subseteq V$ such that $f(X) = t^* a(X)$ and $a(X) > 0$.

By running $\mathsf{COV}(0)$ we can tell whether $t^* = 0$ or $t^* > 0$. So we can assume that $t^* > 0$. If we knew the value of t^* and run $\text{L-COV}(t^*)$, then it would return "$t^* = t^*$" and a subset $X \subseteq V$ such that $f(X) = t^* a(X)$ and $a(X) > 0$, that is, $t^* = f(X)/a(X)$. We try to run $\text{L-COV}(t^*)$ *without knowing* the value of t^*. If we can run $\text{L-COV}(t^*)$ successfully without knowing the value of t^*, we can obtain t^* by $f(X)/a(X)$ using $X \subseteq V$ such that $f(X) = t^* a(X)$ and $a(X) > 0$. The point is *how* to run $\text{L-COV}(t^*)$ successfully without knowing the value of t^*. To achieve this goal, we use Megiddo's parametric search method [15,16].

5.2. Megiddo's parametric search

We give a strongly polynomial time algorithm for Problem LSSP using the parametric search technique of Megiddo [15,16]. We explain this technique in the following paragraphs.

Operations used in running $\text{L-COV}(t^*)$ are additions, subtractions, comparisons, oracle calls for function evaluation of f, and only n multiplications to obtain $t^* a(v)$ for each $v \in V$. So each value which appears in running $\text{L-COV}(t^*)$ can be represented as the form $p - qt^*$ where values p, q are known values and not functions of t^*. We consider trying to run $\text{L-COV}(t^*)$ without knowing the value of t^* with all the values represented as linear functions of t^*. When values are represented as linear functions of t^*, each operation is done as follows:

Operation

An addition: $(p_1 - t^* q_1) + (p_2 - t^* q_2) := (p_1 + p_2) - t^*(q_1 + q_2)$.

A subtraction: $(p_1 - t^* q_1) - (p_2 - t^* q_2) := (p_1 - p_2) - t^*(q_1 - q_2)$.

$$\text{A comparison: } (p_1 - t^*q_1) \overset{?}{\gtreqless} (p_2 - t^*q_2) := \begin{array}{c} (p_1 - t^*q_1) > (p_2 - t^*q_2) \\ \text{or} \\ (p_1 - t^*q_1) = (p_2 - t^*q_2) \\ \text{or} \\ (p_1 - t^*q_1) < (p_2 - t^*q_2). \end{array}$$

An addition of two linear functions of t^* needs 2 scalar additions. A subtraction of two linear functions of t^* needs 2 scalar subtractions. So, even though t^* is not known, additions and subtractions do not change the asymptotic running time of the procedure. A comparison of two linear functions of t^*, however, is not so easy. We consider comparing two linear functions of t^*. Let p_1, p_2, q_1, q_2 be known values. Let us consider the comparison of $p_1 - t^*q_1$ and $p_2 - t^*q_2$. Setting $p = p_1 - p_2,\ q = q_1 - q_2$, we want to decide whether $p - t^*q > 0$, $p - t^*q = 0$ or $p - t^*q < 0$. Now we assume $t^* > 0$. A comparison of $p - t^*q$ can be resolved either immediately, if $pq \leq 0$, or by running Procedure COV with parameter p/q to compare p/q with t^*. We describe below Algorithm LSSP, which solves Problem LSSP within Megiddo's parametric search method.

Algorithm LSSP

Step 1: Decide whether "$t^* = 0$" or "$t^* > 0$" by running COV(0). If $t^* = 0$, then stop.
Step 2: Run L-COV(t^*) without knowing the value of t^* with all the values represented as linear functions of t^*. Each comparison of two linear functions of t^* encountered during the computation can be evaluated (if necessary) by running Procedure COV. We can obtain $X \subseteq V$ such that $f(X) = t^*a(X)$ and $a(X) > 0$.
Step 3: Return $t^* := f(X)/a(X)$.

We will show that Algorithm LSSP solves Problem LSSP in strongly polynomial time after describing two procedures; Procedure COV and Procedure L-COV.

5.3. Comparison of t with t^*

As a preparation for describing Procedure COV and Procedure L-COV, we introduce Algorithm MFM. Let U be a finite nonempty set and $\mathcal{D} \subseteq 2^U$ be a ring family. For a vector $b \in \mathbf{R}^U$, we define a modular function $b_{\mathcal{D}} : \mathcal{D} \to \mathbf{R}$ as

$$b_{\mathcal{D}}(X) = b(X)(X \in \mathcal{D}).$$

Let us consider minimizing a modular function $b_{\mathcal{D}} : \mathcal{D} \to \mathbf{R}$. We assume that a directed graph representation of $\mathcal{D}$ is known. Using a result of Picard [18] modular function minimization over a ring family can be reduced to the minimum cut problem of a network with $O(|U|)$ vertices in $O(|U|^2)$ time, and Cunningham [3] showed the equivalence between the modular function minimization problem and the minimum cut problem. For the minimum cut problem, many combinatorial strongly polynomial time algorithms are known [1], and most of them are fully combinatorial. So we can design a fully combinatorial strongly polynomial time algorithm for modular function minimization over ring families. For example, $b_{\mathcal{D}}$ can be minimized with $O(|U|^3)$ fully combinatorial operations. For a vector $b \in \mathbf{R}^U$ and a ring family $\mathcal{D} \subseteq 2^U$ (we know a directed graph representation of $\mathcal{D}$), let Algorithm MFM be an algorithm which finds a minimizer of a modular function $b_{\mathcal{D}}$ with $\mathcal{T}_{\mathsf{MFM}}(|U|)$ fully combinatorial operations, where $\mathcal{T}_{\mathsf{MFM}}(|U|)$ is some polynomial in $|U|$. We assume $\mathcal{T}_{\mathsf{MFM}}(|U|) = O(\mathcal{T}^{\mathsf{A}}(|U|))$ and $\mathcal{T}_{\mathsf{MFM}}(|U|) = O(\mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(|U|))$.

We describe below Procedure COV, which decides, for any given nonnegative value $t \geq 0$, whether "$t < t^*$", "$t = t^*$" or "$t > t^*$" using conditions (10) directly. In Step 1, we examine whether $ta \in \mathbf{P}(f)$ or not. In Step 2, we maximize $a_{\mathcal{D}_f(ta)}$ and examine whether $t = t^*$ or $t < t^*$.

Procedure COV (Comparison with the Optimal Value)

Input: A nonnegative value $t \geq 0$.
Output: A decision whether "$t < t^*$", "$t = t^*$" or "$t > t^*$".
Step 1: Minimize f_{ta} on 2^V by running $\mathsf{SFM}_{\mathsf{am}}(f_{ta})$. If $\min\{f_{ta}(X) \mid X \subseteq V\} < 0$ then stop ($t > t^*$). (If $\min\{f_{ta}(X) \mid X \subseteq V\} = 0$ we obtain a directed graph representation of $\mathcal{D}_f(ta)$.)
Step 2: Maximize $a_{\mathcal{D}_f(ta)} : \mathcal{D}_f(ta) \to \mathbf{R}$ by running MFM$(-a, \mathcal{D}_f(ta))$. If $\max\{a(X) \mid X \in \mathcal{D}_f(ta)\} = 0$ then stop ($t < t^*$). If $\max\{a(X) \mid X \in \mathcal{D}_f(ta)\} > 0$ then return a maximizer of $a_{\mathcal{D}_f(ta)}$ and stop ($t = t^*$).

As we assumed $n = O(\gamma(f))$ (see Section 4), for any $x \in \mathbf{R}^V$, the function value of f_x can be acquired in $O(\gamma(f))$ time. So the running time of Step 1 is $O(\mathcal{T}(n, \gamma(f)))$. Thus, the total running time of Procedure COV is $O(\mathcal{T}(n, \gamma(f)) + \mathcal{T}_{\mathsf{MFM}}(n)) = O(\mathcal{T}(n, \gamma(f)))$. Let $\mathcal{T}^{\mathsf{O}}_{\mathsf{COV}}(n) = \mathcal{T}^{\mathsf{O}}(n)$, $\mathcal{T}^{\mathsf{A}}_{\mathsf{COV}}(n) = \mathcal{T}^{\mathsf{A}}(n)$, and $\mathcal{T}_{\mathsf{COV}}(n, \gamma(f)) = \mathcal{T}(n, \gamma(f))$.

Let Procedure L-COV be a procedure which is obtained by replacing Algorithm $\mathsf{SFM}_{\mathsf{am}}$ by Algorithm $\mathsf{FC\text{-}SFM}_{\mathsf{am}}$ in Procedure COV. And moreover if $t = t^*$, Procedure L-COV returns a subset $X \subseteq V$ such that $f(X) = t^* a(X)$ and $a(X) > 0$. For any given $t \geq 0$, once $ta(v)$ is computed for each $v \in V$, Procedure L-COV compares t to t^* with $O(\mathcal{T}^{\mathsf{O}}_{\mathsf{FC}}(n))$ oracle calls for function evaluation of f, $O(\mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(n))$ comparisons, and $O(n\mathcal{T}^{\mathsf{O}}_{\mathsf{FC}}(n) + \mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(n))$ additions and subtractions. Let $\mathcal{T}^{\mathsf{O}}_{\text{L-COV}}(n) = \mathcal{T}^{\mathsf{O}}_{\mathsf{FC}}(n)$, $\mathcal{T}^{\mathsf{FC}}_{\text{L-COV}}(n) = \mathcal{T}^{\mathsf{FC}}_{\mathsf{FC}}(n)$, and $\mathcal{T}_{\text{L-COV}}(n, \gamma(f)) = \mathcal{T}_{\mathsf{FC}}(n, \gamma(f))$. The running time of L-COV(t) is $O(\mathcal{T}_{\text{L-COV}}(n, \gamma(f)))$.

5.4. Complexity

The following theorem is the main result in the paper.

Theorem 5.1. *Algorithm* LSSP *solves Problem* LSSP *in strongly polynomial time.*

Proof. The running time in Step 1 is $O(\mathcal{T}_{\mathsf{COV}}(n, \gamma(f)))$. In Step 2, $O(\mathcal{T}^{\mathsf{FC}}_{\text{L-COV}}(n))$ comparisons of linear functions of t^* are evaluated and the running time of the other part is $O(\mathcal{T}_{\text{L-COV}}(n, \gamma(f)))$. So Algorithm LSSP runs in $O(\mathcal{T}^{\mathsf{FC}}_{\text{L-COV}}(n) \cdot \mathcal{T}_{\mathsf{COV}}(n, \gamma(f)) + \mathcal{T}_{\text{L-COV}}(n, \gamma(f)))$ time. □

6. The minimum-ratio problem

Let $f : 2^V \to \mathbf{R}$ be a submodular function with $f(\emptyset) = 0$ and $f' : 2^V \to \mathbf{R}$ be a supermodular function with $f'(\emptyset) = 0$. We assume that $f(X) \geq 0$ for all $X \subseteq V$ and $f'(X) > 0$ for some $X \subseteq V$. Let us consider the following minimum-ratio problem.

Problem MR: Minimize $\dfrac{f(X)}{f'(X)}$ subject to $X \subseteq V$, $f'(X) > 0$.

We give a strongly polynomial time algorithm for Problem MR. We can easily see that Problem MR is equivalent to the following maximization problem of a parameter t.

Problem P-MR: Find $t^* = \max\{t \in \mathbf{R} \mid f(X) - tf'(X) \geq 0, \forall X \subseteq V\}$,
and find $X \subseteq V$ such that $f(X) - tf'(X) = 0$ and $f'(X) > 0$.

Problem P-MR is a generalization of Problem LSSP. The optimal value t^* of Problem P-MR is nonnegative. In the same way as the discussion in Section 2, we have the following conditions for t^* and any $t \geq 0$:

$$\begin{aligned}
t < t^* &\iff \begin{cases} \min\{f(X) - tf'(X) \mid X \subseteq V\} = 0, \\ \max\{f'(X) \mid X \in \arg\min(f - tf')\} = 0, \end{cases} \\
t = t^* &\iff \begin{cases} \min\{f(X) - tf'(X) \mid X \subseteq V\} = 0, \\ \max\{f'(X) \mid X \in \arg\min(f - tf')\} > 0, \end{cases} \\
t > t^* &\iff \min\{f(X) - tf'(X) \mid X \subseteq V\} < 0.
\end{aligned} \tag{16}$$

For any $t \geq 0$, $f - tf'$ is a submodular function defined on 2^V. Using (16) and the same technique as the algorithm for Problem LSSP (Section 5), we can develop a strongly polynomial time algorithm for Problem P-MR and simultaneously for Problem MR. Note that a supermodular function maximization problem on a ring family $\mathcal{D}$, or a submodular function minimization problem on $\mathcal{D}$, can be reduced to a normal submodular function minimization problem if we know a directed graph representation of $\mathcal{D}$. (See Schrijver [21].)

Acknowledgments

I would like to thank Satoru Iwata for a number of useful suggestions, including the efficient method of constructing all the minimizers of a submodular function via the IFF algorithm, and two anonymous referees for their helpful comments on an earlier draft of this paper.

References

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows—Theory, Algorithms, and Applications, Prentice-Hall, Englewood, NJ, 1993.
[2] R.E. Bixby, W.H. Cunningham, D.M. Topkis, The partial order of a polymatroid extreme point, Mathematics of Operations Research 10 (1985) 367–378.
[3] W.H. Cunningham, Minimum cuts, modular functions, and matroid polyhedra, Networks 15 (1985) 205–215.
[4] J. Edmonds, Submodular functions, matroids, and certain polyhedra, in: R. Guy, H. Hanai, N. Sauer, J. Schönheim (Eds.), Combinatorial Structures and Their Applications, Gordon and Breach, New York, 1970, pp. 69–87.
[5] L. Fleischer, S. Iwata, A push-relabel framework for submodular function minimization and applications to parametric optimization, Discrete Applied Mathematics 131 (2003) 311–322.
[6] S. Fujishige, Lexicographically optimal base of a polymatroid with respect to a weight vector, Mathematics of Operations Research 5 (1980) 186–196.
[7] S. Fujishige, Submodular Functions and Optimization, North-Holland, Amsterdam, 1991.
[8] D. Gale, A theorem on flows in networks, Pacific Journal of Mathematics 7 (1957) 1073–1082.
[9] D. Hartvigsen, A submodular optimization problem with side constraints, Mathematics of Operations Research 23 (1998) 661–679.
[10] D. Hartvigsen, A strongly polynomial time algorithm for a constrained submodular optimization problem, Discrete Applied Mathematics 113 (2001) 183–194.
[11] S. Iwata, A fully combinatorial algorithm for submodular function minimization, Journal of Combinatorial Theory (B) 84 (2002) 203–212.
[12] S. Iwata, A faster scaling algorithm for minimizing submodular functions, SIAM Journal on Computing 32 (2003) 833–840.
[13] S. Iwata, L. Fleischer, S. Fujishige, A combinatorial strongly polynomial algorithm for minimizing submodular functions, Journal of the ACM 48 (2001) 761–777.
[14] N. Megiddo, Optimal flows in networks with multiple sources and sinks, Mathematical Programming 7 (1974) 97–107.
[15] N. Megiddo, Combinatorial optimization with rational objective functions, Mathematics of Operations Research 4 (1979) 414–424.
[16] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, Journal of the ACM 30 (1983) 852–865.
[17] K. Murota, Discrete Convex Analysis, Society for Industrial and Applied Mathematics, Philadelphia, 2003.
[18] J.C. Picard, Maximal closure of a graph and applications to combinatorial problems, Management Science 22 (1976) 1268–1272.
[19] T. Radzik, Parametric flows, weighted means of cuts, and fractional combinatorial optimization, in: P.M. Pardalos (Ed.), Complexity in Numerical Optimization, World Scientific, Singapore, 1993, pp. 351–386.
[20] T. Radzik, Fractional combinatorial optimization, in: D.Z. Du, P.M. Pardalos (Eds.), in: Handbook of Combinatorial Optimization, vol. 1, Kluwer Academic Publishers, Boston, 1998, pp. 429–478.
[21] A. Schrijver, A combinatorial algorithm minimizing submodular functions in strongly polynomial time, Journal of Combinatorial Theory (B) 80 (2000) 346–355.
[22] J. Vygen, A note on Schrijver's submodular function minimization algorithm, Journal of Combinatorial Theory (B) 88 (2003) 399–402.